PATENT APPLICATION
DOCKET NO.: 200209135-1

"EXPRESS MAIL" Mailing Label No..EV331251418US
Date of Deposit.....OCTOBER 1, 2003.....

SYSTEM AND METHOD FOR TESTING A CIRCUIT DESIGN

CROSS-REFERENCE TO RELATED APPLICATION(S)

[0001] This application discloses subject matter related to the subject matter disclosed in the following commonly owned co-pending patent applications: "System and Method for Generating a Test Case," filed _____, Application No.: _____ (Docket Number 200209280-1), in the names of Ryan C. Thompson, John W. Maly, and Zachary S. Smith; and "System and Method for Building a Test Case Including A Summary of Instructions," filed _____, Application No.:_____ (Docket Number 200208930-1), in the names of Ryan C. Thompson, John W. Maly, and Adam C. Brown, both of which are hereby incorporated by reference for all purposes.

BACKGROUND

[0002] The design cycle for a digital integrated circuit is long and expensive. Once the first chip is built, it is very difficult, and often impossible, to debug it by probing internal connections or to change the gates and interconnections. Usually, modifications must be made in the

original design database and a new chip must be manufactured to incorporate the required changes. Since this process can take months to complete, chip designers are highly motivated to attempt to perfect the chip prior to manufacturing.

[0003] It is therefore essential to identify and quantify the architectural requirements necessary to assure good performance over a wide range of events prior to manufacturing the digital integrated circuit. Accordingly, a simulator comprising program code may be employed to provide a simulation environment that is executable on top of a host computer platform in order to model at least some or all of the functionalities of the desired integrated circuit, for example, a target processor core. The characteristics of the target processor core that the simulator emulates may be specified to include processor architectural model, processor clock speed, cache configuration, disk seek time, memory system bus bandwidth, and numerous other parameters. The resulting software-based target processor core provides the appearance of being a normal processor core while a test generator exercises the target processor core with test cases in order to collect detailed hardware behavior data only available through the use of the simulation. In particular, inputs supplied the test generator define specific hardware functionalities to be tested. The resulting test files the test generator gathers may be subsequently utilized to better understand various aspects of the simulated target processor's core.

[0004] By modeling the processor core, simulation is able to provide an accurate behavioral paradigm that allows each

aspect of the simulated processor core's functionality to match that of a specific target processor core. As a result, simulations can provide visibility that translates into detailed information regarding the various aspects of a target processor core's execution behavior. Simulators are not without limitations, however. Test generators often generate test cases containing illegal test behavior and require debugging. As a result, once the test generator has been debugged, the inputs provided the test generator must be laboriously reconstructed so that the target processor core may be retested.

SUMMARY

[0005] A system and method are disclosed that provide for testing a circuit design using a test generator. In one embodiment, a random number generator, responsive to a seed, generates a random number sequence and an event probability generator, responsive to profile settings, generates a probability profile. The test generator, responsive to the random number sequence and the probability profile, generates a test case that includes settings indicative of the seed and the profile settings. An extraction and regeneration engine extracts the seed and the profile settings from the test case in order to reconstitute a test case designed to avoid illegal test behavior.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] FIG. 1 depicts a block diagram of an embodiment of a system for simulating target processor core models that can be exercised by an Automatic Test Generator (ATG);

[0007] FIG. 2 depicts a block diagram of an embodiment of a system for testing a circuit design using an ATG;

[0008] FIG. 3 depicts a block diagram of an embodiment of an extraction and regeneration engine operable to extract data from a test structure in order to reconstruct test case inputs;

[0009] FIG. 4 depicts a flow chart of an embodiment of a method of testing a circuit design using an ATG; and

[0010] FIG. 5 depicts a flow chart of one embodiment of a method of debugging an ATG.

DETAILED DESCRIPTION OF THE DRAWINGS

[0011] In the drawings, like or similar elements are designated with identical reference numerals throughout the several views thereof, and the various elements depicted are not necessarily drawn to scale. Referring now to FIG. 1, therein is depicted an embodiment of a system 100 for simulating target processor core models that can be exercised by an Automatic Test Generator (ATG) 102. A host platform 104 includes a host OS 106 executing thereon that is operable to provide a software platform. A simulator environment 108 is provided as a software rendition capable of running on the host OS 106, and may be embodied as one or more simulated target instances that define a simulated environment. As illustrated, the simulator environment includes a Register-

Transfer Level (RTL) model 110 of the target processor core and an architectural simulator model 112 of the target processor core. As will be explained in further detail hereinbelow, the ATG 102, responsive to input settings, generates a test case that exercises the behaviors and functionalities of the RTL model 110 and the architectural simulator model 112. The resulting test files are then utilized to understand the behavior of the target processor core.

[0012] The RTL model 110 and the architectural simulator model 112 may simulate any processor core having any configuration or digital design. For example, the target processor core may simulate a two-core processor system wherein each processor is capable of parallel instruction processing, i.e., multi-threading, that delivers high availability and scalability with a wide breadth of enterprise application capabilities. It should be appreciated that depending on the design and verification objectives, the simulator environment 108 may include other types of target processor core models.

[0013] The RTL model 110 simulates the target processor core by utilizing a hardware-description language (HDL) that specifies the signal and gate-level behavior of the target processor core. The architectural simulator model 112, on the other hand, provides a higher level of abstraction than the RTL simulator model 110 in order to model the target processor core in terms of a high-level architecture that defines system-level behavioral functionalities of the target processor core. The RTL model 110 may be designed with the aid of computer-aided design (CAD) software tools, also

referred to as computer-aided engineering (CAE) software tools, that assist in the development of the conceptual and physical design of the IC as well as the verification of the IC.

[0014] Sophisticated CAD software tools contain component libraries and component models that describe in detail the logical and electrical operations of the digital system design of the IC. Using these models, the IC design may be verified so that various types of logic and timing errors may be found by the test generator during the pre-silicon simulation phase of development. Specifically, the RTL model 110 may be designed by a schematic editor in a highly capable HDL environment such as a Very High Speed Integrated Circuit (VHSIC) hardware description language (VHDL) environment, a Verilog description language environment, or an Advanced Boolean Equation Language (ABEL) environment, for example. The HDL language environment provides a design, simulation, and synthesis platform wherein each constituent component within the design can be provided with both a well-defined interface for connecting it to other components and a precise behavioral specification that enables simulation. The architectural model 112, on the other hand, may be implemented in a higher-level language such as C or C++.

[0015] FIG. 2 depicts a system 200 for testing a circuit design using an ATG 202 according to one embodiment of the invention. A random number generator 204, responsive to a seed 206, generates a random number sequence 208. In one embodiment, if the seed is a number A, the random number generator 204 generates the random number sequence 208 $\{A_1,$

A_2, A_3, ..., A_n}. By way of another example, if the seed 206 provided to the random number generator 204 is B, then the random number sequence 208 generated is {B_1, B_2, B_3, ..., B_m}. Hence, the random number sequence 208 may be considered a function of the seed 206. In another embodiment, the random number sequence 208 may be considered a random sequence of numbers that are "predetermined" based upon the seed 206.

[0016] An event probability generator 210, responsive to profile settings 212, generates a probability profile 214. The profile settings 212 are user configurable settings which define the frequency or occurrence of the events that will exercise the processor models. Events include data operations such as loading, storing, and arithmetic operations, for example. Moreover, events include other performance-based operations such as the selection of parameters related to floating-point representations of numbers. The event probability generator 210 reconstitutes the profile settings 212 into the probability profile 214 which defines a set of event-related parametrics that will be accepted by the ATG 202.

[0017] The ATG 202, responsive to the random number sequence 208 and the probability profile 204, generates a test case 216 in order to exercise the processor models 218. Additionally, command line settings may be provided to the ATG 202 and employed by the ATG 202 in the generation of the test case 216. Command line settings relate to the starting and stopping of specific actions in the processor models 218 and the defining of testing conditions, such as the number of threads, for example. In one embodiment, the ATG 202

embeds settings indicative of the seed 206, the profile settings 212, and the command line settings within the test case 216. Further information regarding the generation of test cases, especially those involving multithreading, may be found in the aforementioned patent application entitled "System and Method for Generating a Test Case," filed _____, Application No.: _____ (Docket Number 200209280-1), in the names of Ryan C. Thompson, John W. Maly, and Zachary S. Smith, which is hereby incorporated by reference for all purposes.

[0018] As illustrated, the test case 216 exercises an RTL model 220 and an architectural simulator model 222. The results of the exercises are stored as test files 224. Included in the data in the test files 224 are settings indicative of the seed 206, the profile settings 212, and the command line settings. As will be discussed in more detail hereinbelow, the embedded settings in both the test case 216 and the test files 224 are indicative of the seed 206, the profile settings 212, and the command line settings. Either the test case 216 or the test files 224 may provide an extraction and regeneration engine the information necessary to reconstruct the inputs supplied to the ATG 202 so that the ATG 202 may be debugged or a modified ATG can rerun the test with the identical inputs. A comparator 226, which may be a programmer, a group of programmers, an expert system, or any combination thereof, examines the content of test files 224 to determine if the test results are valid or invalid. In particular, the comparator 226 examines and compares the results provided by the RTL model 220 and the results provided by the architectural simulator model 222. As

previously discussed, the RTL model 220 simulates register-level events in the target processor core. The RTL model, therefore, serves as a verification tool for the architectural simulator 222. Additionally, the comparator 226 examines the output provided by the RTL model 220 and the architectural simulator 222 to determine if an illegal test behavior has occurred.

[0019] If the test files are valid, i.e., the RTL model 220 verifies the architectural simulator model 222 and the test files 224 do not contain illegal test behavior, the test files 224 become valid test results 228 which provide detailed information regarding each exercised aspect of the target processor core's execution behavior. On the other hand, if the test files 224 indicate processor model inconsistences between the RTL model 220 and architectural simulator 222, then a debugging operation 230 may be required with respect to the processor models. Debugging the architectural simulator and RTL models may involve diagnosing and resolving the problem according to conventional techniques. For example, by examining the test files 224 and underlying HDL-based code of the RTL model 220, a clear understanding of the symptoms of the problem may be achieved. Then all of the variables that affect the problem may be identified and the variables progressively eliminated until the root cause of the problem is isolated. Once the root cause is isolated, the HDL-based code of the models may be appropriately modified to eliminate the problem. If the test files 224 indicate the presence of an illegal test behavior, however, the ATG 202 requires a debugging operation 232 which will be described more fully with particular reference to

FIG. 3 hereinbelow. An illegal test behavior may involve a request to perform an illegal operation such as the simultaneous storing and reading of data in a particular register. For example, the ATG may incorrectly instruct the test case to read data at a particular register while the data is being stored at that register, thereby creating an illegal test behavior.

[0020] FIG. 3 depicts one embodiment of a parametric extraction and regeneration system 300 for effectuating the ATG debugging operation 232 illustrated in FIG. 2. In order to troubleshoot the ATG 202 that has created an illegal test behavior, the comparator 226, e.g., the programmer or group of programmers, systematically examines a test structure 301 which may be the test case 216 in one embodiment or the test files 224 in another embodiment. As illustrated, test structure 301 include random seed settings 302, instructions 304, profile settings 306, command line settings 308, and instructions 310, for example. In particular, with respect to performing debugging operations 311 on the ATG 202, the comparator 226 will examine the test structure in detail in order to facilitate the generation of reconstituted test cases based on extracted test case parametrics. The troubleshooting techniques employed to determine the cause of the illegal test behavior may be similar to the aforementioned techniques employed in relation to debugging the processor models. Additionally, the troubleshooting techniques may involve setting break-points, for example, in order to isolate the code portion responsible for the illegal test behavior. In particular, the debugging operations 311 will involve an extraction and regeneration engine 318

extracting the seed 206, profile settings 212, and command line settings 308 from the test structure 310, which may be the test case 216. As will be explained in more detail hereinbelow, the seed 206, profile settings 212, and command line settings 308 form extraction files 320 which serve as reconstituted input parameters to the random number generator 204 and event probability generator 210 such that the random number sequence 208 and probability profile 214 originally provided may be presented again to the ATG 202 to facilitate its debugging. Once the problem in the ATG is isolated, the comparator 226 is operable to modify or otherwise add applicable functionality to the ATG 202 to correct the cause of the illegal test behavior. The result of the modification, i.e., modified ATG 314, is operable to create a modified test case 316 and exercise the processor models with the modified test case 316 in order to ensure that the illegal test behavior has been eliminated.

[0021] In order to generate the modified test case 316 that corresponds to the original test case, however, the modified ATG 314 must be supplied with the same random number sequence 208 and probability profile 214. The extraction and regeneration engine 318 traverses the test structure 301, which may be the test case 216 or the test files 224, to extract the random seed settings 302, profile settings 306, and command line settings 308. Based on the extracted settings 302, 306, and 308, the extraction and regeneration engine 318 places the seed 206, the profile settings 212, and command line settings 308 in one or more extraction files 320. In one embodiment, the extraction and regeneration engine 318 automatically extracts the necessary inputs for

the modified ATG 314, i.e., the seed 206, profile settings 212, and command line settings 308. The extraction and regeneration engine 318 may be implemented using a suitable software language such as C, C++, or Perl, for example.

[0022] The extraction files 320 are employed by the random number generator 204 and the event probability generator 210 to generate the random number sequence 208 and probability profile 214, respectively, required to generate the modified test case 316. In particular, the seed 206 supplied to the random number generator 204 is the same seed 206 supplied to the random number generator 204 in FIG. 2. For example, if the seed 206 used to ultimately generate the test case 216 of FIG. 2 was equal to A, then the seed 206 used to ultimately generate the modified test case 316 is equal to A. Continuing with this example, the random number sequence 208 generated by the random number generator 204 in FIG. 2 would have been $\{A_1, A_2, A_3, \ldots, A_n\}$ and the random number generator 204 generates the same random number sequence 208 $\{A_1, A_2, A_3, \ldots, A_n\}$ using the extracted seed. Similarly, the profile settings 212 and command line settings 308 supplied to the modified ATG in order to generate the modified test case 316 are identical to the profile settings 212 and the command line settings 308 previously used to generate the test case 216. Therefore, the systems and methods disclosed herein provide the modified ATG 314 the same inputs as the ATG 202 in order to retest the processor models 218 with the modified test case 316 which is identical to the test case 216, but for the modifications made in order to eliminate the illegal test behavior. In particular, the ATG 202 cooperates with the extraction and regeneration

engine 318 to provide this automated scheme of testing. As explained previously, the ATG 202 operates responsive to settings indicative of the seed 206, profile settings 212, and command line settings 308 in the test case 216 so that the settings will be embedded in the test files 224. The extraction regeneration engine 318 can then automatically extract and reconstruct the seed 206, profile settings 212, and command line settings 308 from settings embedded in the test structure 310. With this scheme, the systems and methods disclosed herein provide for efficient circuit design testing and minimize the labor associated with reconstructing the inputs originally supplied to the ATG 202.

[0023] By way of implementation, if an illegal test case is called *test100.tc* and the new, modified version of the ATG is located at */usr/bin/testgen*, then the automatic extraction and regeneration process may be effectuated by calling out a script, e.g., "*regen*," and executing a modified test case as follows:

regen test100.tc /usr/bin/testgen

As may be appreciated by those skilled in the art, additional software functionalities can also be provided (for example, via software switches, dynamically linkable module options, and the like) to customize or further modify the dynamics of the extraction and regeneration process described herein. It should be appreciated from the foregoing description of FIG. 3 that the extraction and regeneration engine may be utilized for multiple purposes relating to reconstructing

test case input parameters. Specifically, the extraction and regeneration engine may be employed to debug the ATG by automatically extracting and reconstituting the test input parameters, i.e., the seed and the profile settings, so that the ATG may be debugged. Additionally, the extraction and regeneration engine may be employed to supply a modified ATG, i.e., an ATG modified to remove illegal test behavior, reconstructed input data so that a modified test case may be generated for re-exercising the processor models. The following Figures, FIG. 4 and FIG. 5, illustrate these two aspects in additional detail.

[0024] FIG. 4 depicts a method of testing a circuit design using a test generator according to one embodiment. At block 400, an illegal test behavior is detected in a test file that is produced by exercising a test case generated by the test generator on a model of the circuit design. At block 402, profile settings are extracted from a test structure relating to the test case, e.g., the test file or the test case itself. At block 404, a random number seed is extracted from the test structure. Additional information such as command line settings may also be extracted from the test structure. It should be appreciated that the extraction operations of blocks 402 and 404 may occur in any order or simultaneously. Moreover, the extraction operations of blocks 402 and 404 may occur automatically in order to maximize the efficiency of testing operations. At block 406, input data supplied to the test generator is reconstructed from the profile settings and the random number seed. At block 408, the reconstructed input data is supplied to a modified test generator that has been modified to avoid illegal test behavior.

[0025] FIG. 5 depicts one embodiment of a method of debugging a test generator. At block 500, a test case generated by the test generator is verified, for example, by executing the test case on a processor model. At block 502, profile settings are automatically extracted from the test case. At block 504, a random number seed is automatically extracted from the test case. It should be appreciated that the operations of blocks 502 and 504 may be performed in any order or simultaneously. At block 506, test input parameters are automatically reconstituted from the extracted profile settings and the random number seed. Additionally, command line settings may be extracted as a portion of the input parameters. At block 508, the reconstituted input parameters are supplied to the test generator for debugging thereof by a comparator which may be a programmer, a group of programmers, an expert system or any combination thereof.

[0026] Although the invention has been particularly described with reference to certain illustrations, it is to be understood that the forms of the invention shown and described are to be treated as exemplary embodiments only. Various changes, substitutions and modifications can be realized without departing from the spirit and scope of the invention as defined by the appended claims.